

Composite System Design: the Good News and the Bad News

Martin S. Feather*, Stephen Fickas**, B. Robert Helm**

*USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292

**University of Oregon, Eugene, OR 97403

1: Introduction

Composite systems are multi-agent systems. They comprise several agents interacting so as to achieve some system-wide goals or functional requirements. In our experience, composite systems are numerous - it is more difficult to think of systems that are not composite than those that are. Thus we see broad applicability for a composite system design viewpoint in the early stages of requirements elicitation and specification design.

We are attempting to develop knowledge-based assistance for engineering composite systems. In particular, we wish to support the production of formal, operational specifications of composite systems from formal statements of requirements. We would like automated tools to help the analyst to analyze a set of requirements and identify what agents are necessary, what capabilities each agent must have to fulfill its role in the overall system, the inter-agent protocol that allows agents to cooperate, and finally, the interface each agent must have to be an active participant in the system.

The composite system design approach has been proposed by Feather to address this problem [6]. The essence of his composite system design approach is to do the design (or redesign) of composite systems by beginning from a description of the properties desired of the system as a whole, and then deriving the behaviors of, and interactions between, the agents so that their combination will achieve the desired system properties. The final system may include pre-existing agents whose properties cannot be changed, and newly created agents defined just for the task at hand.

Typical systems will be a mixture of human, software, and hardware agents. Taking the standard elevator problem as an example, we might identify elevator controller agents, passenger agents, maintenance agents, etc. In essence, CSD allows us to explore the entire space of designs that might satisfy the system requirements. Choosing among alternative implementations of these

designs will lead us to fully automated elevators, fully manual elevators, and many specifications in between.

In this section we outline how our composite system design work fits with other requirements and specification activity. We then summarize our approach and how we are pursuing its study, and highlight the key questions that we are attempting to answer. The sections that follow explore these issues in more detail:

In section 2 we show how we model the composite system design process as a *search through a space of possible designs*.

In section 3 we focus on the *construction* of the search space, arguing that a relatively small number of domain-independent methods suffice to generate an interesting class of designs.

In section 4 we focus on the *evaluation* of alternative designs, and show how this is related to the additional problems inherent in *evolving existing systems*.

In section 5 we summarize our findings, and outline the areas we feel to be the most worthy of immediate study.

1.1: Relationship to other design activity

Most work on requirements and specification addresses the design of components in relative isolation from the design and/or purpose of the environment of which they are but a part. In other words, when an component (an embedded software system, say) is to be described and thereafter constructed, its environment is typically taken as immutable, and is described only to the minimal extent that is necessary to delineate how the component must behave. Many of the examples used to illustrate requirements and specification work exhibit this characteristic, e.g., the problem set in [11] included:

- Problem #1, LIBRARY ([13]) - a description of the required transactions of a simple book library, but lacking a description of the broader purpose of the system, the sharing of a resource among multiple users;

- Problem #4, LIFT - a description of the required activity of a lift (elevator) system in response to button presses, but again lacking a description of the broader purpose, in this case to assist the transportation of people in a multi-story building to their desired destination floors.

We believe that problem specifications like the two above place unnecessary limitations on the implementer. In reaction to this, our study of composite system design emphasizes the elicitation and study of the larger context of each component or *agent*. Thus we see the *end-product* of our design process as the *starting point* for “traditional” requirements and specification work. We believe the following advantages accrue from this broader scope of consideration:

- *Traceability to system-wide goals.* The system agents and communications among those agents that result from this design process can be traced to the initial system-wide goals, and the design choices made. This provides a formal rationale for the design. In contrast, a description of only the end product of composite system design, namely the behaviors of the individual agents, might not clearly reflect the underlying rationale.
- *Thorough exploration of the space of design alternatives.* By beginning with the system-wide goals as the objective of the design process, we do not inadvertently pre-commit to any particular decomposition of those goals among the agents. This maximizes the likelihood that we do not overlook alternative, perhaps superior, solutions.
- *Basis for redesign.* In the event of the need to make or respond to some change (either to the system-wide goals, to the capabilities of, or communication between, agents of the system, or to the relative cost of design alternatives), the record of the design process will serve as the basis on which to do such redesign. Without such a record, it would be hard to redesign the system in a principled manner.

We believe this last point is particularly crucial. As an example, consider developing the requirements for the elevator controller above. If we treat the controller in isolation from its environment, we cannot formally explain the need for any feature of the elevator, such as the presence of doors or the use of a demand-driven service protocol. Without such rationale, it is difficult to formally prove we can eliminate a given button as an economy measure, or that we should use it in a new elevator installation in another building. Finally, ignoring the high-level goals of passengers and other agents in the elevator system makes it difficult to describe or evaluate innovative designs which new information technology may make feasible, such as an elevator which predicted the arrival

floor of passengers, or one which took voice reservations over a cellular phone. Because of these limitations, we argue that specification approaches which currently take a single-agent, stand-alone view to what are, in reality, composite systems, can benefit from the broader perspective of CSD.

1.2: Research methodology

Our research methodology for studying composite system design has been as follows:

1. Propose a method for composite system design, i. e., a method whose end-product would suffice as the starting point for “traditional” requirements and specification work. This paper summarizes our current method in section 1.3.
2. Identify the crucial research questions to ask. We identify two we feel are most important below.
3. Exercise our approach on a diverse set of design problems, for which well-documented designs exist. We discuss our choice of problems in section 3.0.
4. Evaluate how well we have addressed our research questions in each of these exercises. We summarize these results in the final section.

Example problems we have studied with this research methodology include elevator systems, train systems, air traffic control, libraries, power plant control, e-mail systems, and (most recently) meeting management systems. Of the many research questions we tackled in these exercises, we discuss two in this paper:

1. Is CSD *sufficient* to generate designs for realistic problems?

Our studies indicate that a small set of transformations can construct a diverse range of existing designs.

2. Can CSD be made *tractable* for realistic problems?

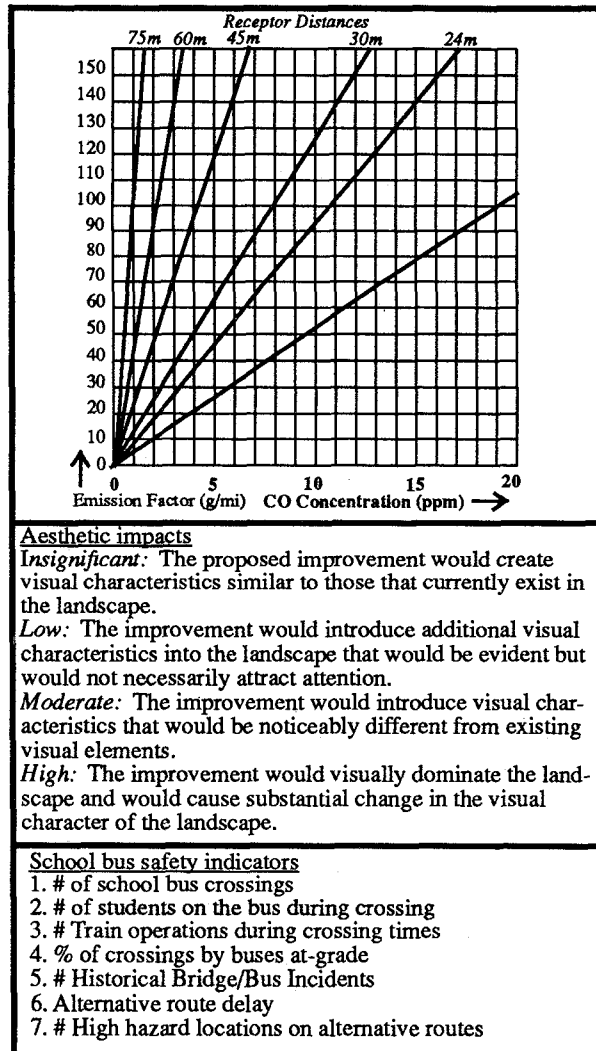
Our studies have helped identify the kinds of domain-specific knowledge we would need to make CSD tractable; however, little research has been done to date on integrating this knowledge into the design process.

While we have formalized some pieces of CSD, and automated still smaller pieces, we view our work to date as only a feasibility study. This paper reports on this study, and concludes with a future work section in which we briefly summarize the principal research problems we see remaining.

1.3: The composite design approach

The starting point for CSD is an initial specification describing system-wide goals (constraints), e.g., “move passengers to their destinations”, and capabilities of the

Figure 4 Operating impact models.



In addition, the specification process of large-scale composite systems must consider the *transition impact* from standards which may exist in the given domain. Agents, whether software, hardware, or human, must be programmed, designed, or trained to carry out a given set of responsibilities. In a system where a large group of agents have similar responsibilities, it is typically costly to alter these responsibilities. In the rail-highway crossing, for instance, we cannot easily change the protocol by which cars cross railroad tracks "at-grade" (on the same level as the track), even if our new protocol would be a major improvement. The cost of training all potential agents (the drivers) of their new responsibilities is likely to be prohibitive. Even altering the format or behavior of the lights, gates, and gongs at crossings would probably entail unreasonable expense in the short term, since it would require selecting custom hardware and software over the cheaper, mass-produced equipment available to support the

standard design. At present, few requirements engineering models attempt to evaluate transition impacts when examining a system.

4: Summary and future work

We return to our state-based search perspective to discuss the results of our study of CSD. There are two search components to consider: 1) the design operators that construct new states, and 2), the evaluation heuristics that guide the search to acceptable solutions (composite system specifications). Our design operators take the form of transformations on goals and agents. We have been able to produce a small number of powerful transformations that apply across the transportation problems that we have studied, and which could serve as the basis of an interactive design assistant. These transformations are interactive because we lack the formal analytic models (e.g., theorem provers) to guarantee the responsibility-assignment operation of CSD. We have partially plugged this analysis gap with tools like OPIE and the RG tool. We also continue to work on automating our transformations, gradually making them less dependent on human intervention.

It is not surprising that evaluation heuristics are crucial to making composite design search tractable, and that integrating these heuristics into the search appears to be a more difficult problem than defining transformations which generate the search space. On the positive side, we have found numerous formal and informal models for evaluation in the domains we have studied. By studying small but *realistic* composite system problems, we have also begun to identify some important types of evaluation - brownfield and greenfield -- that these models perform, and the kinds of knowledge needed for these model types.

On the negative side, there has been little work in the requirements and specification field to address the issues of integrating greenfield and brownfield models into the design process. One immediate gain would be to integrate existing evaluation models into a tool based on CSD. This is a task we have taken on in a tool we call Critter [9]. Our initial goal is to *informally* catalog the type of models discussed in section 4. Of course, the actual integration of these models into an *automated* search-based design tool is a difficult task, indeed - it requires mapping between multiple ontologies at various levels of formalism. While Farley&Liu have shown that this is possible in non-composite domains [5], we believe much hard work lies ahead to scale their results up.

5: Acknowledgments

Feather has been supported in part by Defense Advanced Research Projects Agency grant No. NCC-2-

pre-existing agents of the system, e.g., “an elevator can move to an adjacent floor; a passenger at a floor can enter an open-doored elevator at that same floor”. These amount to the functional requirements of the system.

The design process proceeds by incrementally assigning goals as the *responsibility* of subsets of agents: only those agents responsible for a goal are expected to limit their own behavior to ensure satisfaction of that goal. For example, if the elevator system alone is responsible for keeping passengers from falling down elevator shafts, then the elevator must keep doors closed when necessary rather than rely upon passengers to limit their choice of when to walk through an open doorway. Different responsibility assignments lead to radically different systems. For instance, express elevators, scheduled elevators, reserved elevators, prison elevators, freight elevators, etc., all can be generated by exploring alternative responsibility assignments.

During the course of design, the interaction between agents - communication and control - is established. Such interactions may necessitate the introduction of, say, communication media, protocols for communication, and even further agents to facilitate communication. The introduction of these is motivated and rationalized in terms of the overall design process.

This design process ends when all goals have been subdivided and assigned as the responsibility of individual agents, at which point those agents can be independently implemented, assured that their combination will achieve the system-wide goals of the composite system to which they belong. In effect, the system goals have been “compiled out” of the global specification into the individual agent specifications. It is possible that each agent could be a smaller composite system, in which case this design process may be recursively applied. For instance, an entire elevator composite system may be a single agent of a larger transportation composite system, e.g., within a train station or airport.

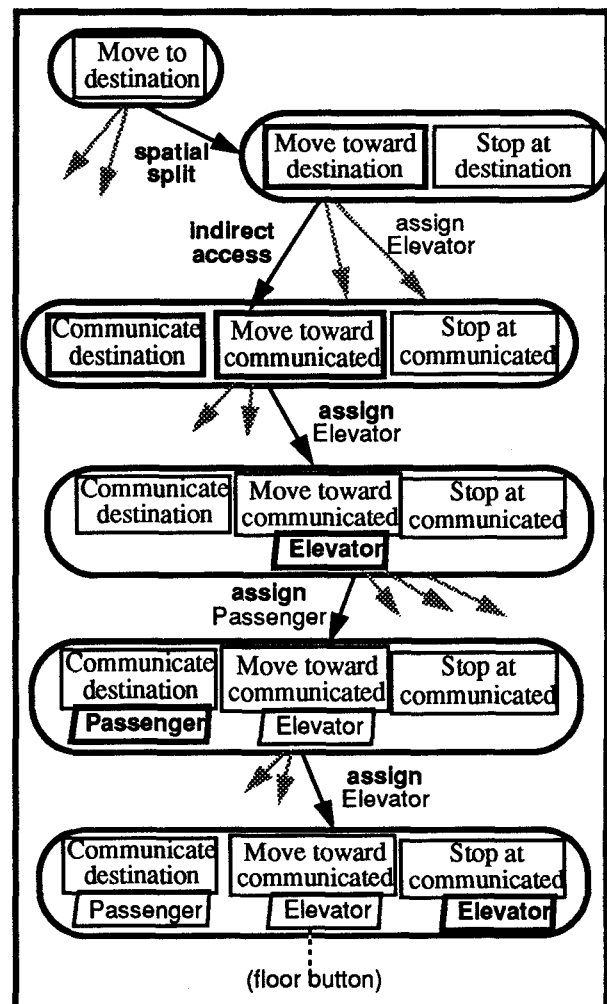
The end product of this process is (a) the specification of individual agent behaviors, and (b) a history of the development that rationalizes those behaviors in terms of the system goals. Figure 1 (adapted from [9]) shows part of the end product of CSD applied to the elevator problem. Starting from a system goal (“move passengers to their destinations”), we derive the responsibility of each passenger agent to notify the elevator of its destination, while the elevator receives the responsibility to move to that destination when it is known. Along the way, we derive the need for an interface component (implemented by a button) which allows the passenger to communicate a destination floor to the elevator. The system specification which results specifies the individual behavior of elevator and passenger agents, but these behaviors combine to give

a specified global behavior. The history of the design process also partially captures alternative design paths, which lead to radically different systems.

2: Searching the CSD space

We formalize the composite system design process by treating it as a search through the space of possible designs. The “states” of this space are designs or specifications; the “operators” are transformations or “methods” [8] which map from one design state to the next. Figure 1 shows part of the state space for the elevator example discussed above. A key question is: is the CSD approach sufficient to *construct* a reasonably complete design space for realistic problems?

Figure 1 Elevator design history (except).

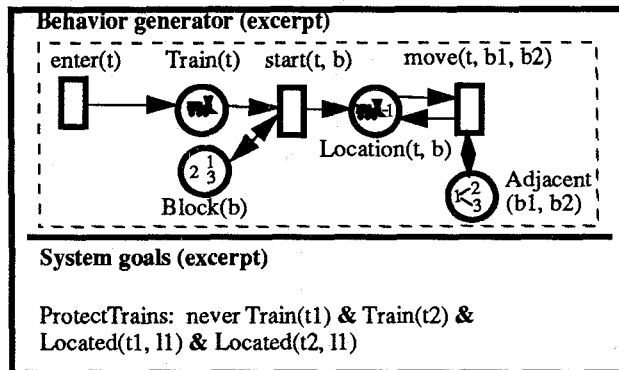


We begin by briefly summarizing the components of the states in the CSD search space, which we discuss in greater detail in [9]. Each state has two components:

1. A behavior *generator* that computes all possible interleavings of the behaviors of individual agents in the system. For the purposes of this paper we assume the behavior generator is written in a discrete event language that can be viewed, alternatively, as a subset of Gist [14] or a high level Petri Net [20],[10].
2. A set of constraints or *goals* which describe restrictions on the behavior desired of the system. The goal/constraint language is a form of temporal logic, roughly similar to that of that of the ERAE language [4] and of the distributed-action logic of [3].

As an example, Figure 2 shows part of the initial state for a train system control problem. To summarize, the part of the behavior generator shown indicates that trains can enter the system, and can (non-deterministically) move among the adjacent locations (blocks) of the train system. The goal shown under "system goals" is a safety constraint: two trains should never occupy the same block.

Figure 2 Excerpt of train example specification.



A solution state for this search process is a specification where all behaviors produced by the behavior generator meet the system goals. In such designs, we have successfully pushed the goals into the individual agent specifications. Our approach to detecting solution states is by counterexample: We attempt to show that a given state is *not* a solution by exhibiting a sequence of events produced by the behavior generator which violates one or more system goals. If such a counterexample can be found, we attempt to identify the deficiency in the specification which produced it, and apply design operators to repair the deficiency, (hopefully) moving closer to a solution.

To produce counterexamples from specifications, we have built two analysis tools¹:

1. We see a continuing need for both tools: OPIE provides efficiency through goal-directed search and abstract planning; the RG tool can be costly to run, but provides more powerful forms of analysis.

1. A planner or scenario generator called OPIE [1]. OPIE can be used in two ways: 1) to disprove a constraint by producing a disallowed behavior (i.e., counter-planning), and 2) to prove an existence goal, e.g., there exists at least one behavior that satisfies some predicate.
2. A reachability-graph (RG) tool. The tool first produces a reachability graph from a static analysis of the generative part, and then allows queries about reachable states. As with OPIE, these queries can be used to disprove a constraint or prove existence goals. However, unlike OPIE, the graph can be used, in conjunction with omega values, to disprove temporal goals such as "trains will eventually reach their destination".

As an example of a disproof that either tool could produce, but which we present in OPIE style for readability, consider the specification of Figure 2. Note that Figure 2 is a starting specification. It specifies the environment in which we will design a train system, but does not yet specify any of the agents we will need to meet the goals. In [9], we follow the elaboration of this naive model into its final, complex, composite form.

OPIE is called to disprove the specification, given certain initial conditions:

(U): **Disprove ProtectTrains in SYS0 given Block(b1) & Block(b2) & Adjacent(b1, b2)**

(O): The goal ProtectTrains in SYS0 is violated by scenario S1:

1. given Block (b1)
2. given Block (b2)
3. given Adjacent(b1, b2)
4. enter => Train(t1)
5. start(t1, b1) => Location(t1, b1)
6. move(Location(t1, b1), b2) & Adjacent(b1, b2) => Location(t1, b2)
7. enter => Train(t2)
8. start(t2, b1) => Location(t2, b1)
9. move(Location(t2, b1), b2) & Adjacent(b1, b2) => Location(t2, b2)

Violation: ProtectTrains in SYS0

We might apply several design operators or methods to address this negative scenario. We could, for example, introduce a new agent (an engineer, for instance) with responsibility for the ProtectTrains goal. However, we could also modify the environment (such as the adjacency relation on blocks), or even the ProtectTrains goal itself. This highlights an important difference between our CSD approach and that of Feather's original CSD [6]. In particular, Feather was concerned with specification *implementation* - given the correct set of goals and the correct set of agents, find a division of responsibility

between them which maintains correctness. Thus, Feather would support only one type of design action here: the restriction of one or more existing agents' behavior to exclude the crash scenario. Our use of CSD, on the other hand, is concerned with specification *design*: given an initial set of goals and an existing environment, attempt to assign responsibility. If this fails (because the initial model was incomplete, because the system is unimplementable, because it is too costly), modify the goals, the environment (including agents), or both. The major ramification of this is that we allow goal modifications (e.g., weakening a goal), environment modifications (e.g., change the existing infrastructure, create new agents) as well as responsibility assignment. Feather takes up some of these broader concerns in [7].

2.1: Planes, trains and automobiles

We have attempted to build a design space constructor based on the CSD model. The approach we have taken is to use transformations as design operators, i.e., as the actions that produce new states/specifications in our state-based search model. Our goal was to define a tractable set of transformations that take goals and behavior generator as input, and produce restricted agent actions as output (the essence of responsibility assignment). We expected these transformations to be interactive, relying on the human specifier to do the complex reasoning sometimes necessary to determine agent action and control. Our success criteria involves a comparison with other interactive, assistant-based design systems (e.g., [8], [16]): if we could obtain the same mixture of human/machine interaction as these systems, we would judge our results as a success.

To test our ideas, we looked at several domains. First, we attempted to redesign Feather's elevator system (as reported in [6]) using a transformational approach. The result, as discussed in [9], was that we were able to rederive the elevator with a relatively small number of transformations.

Our other major effort has been to work our way towards a CSD model of Air Traffic Control (ATC). Preliminary to this, we have studied several simpler problems: traffic intersections as a CSD problem; rail-highway crossings as a CSD problem; train/subway control as a CSD problem. Our general approach has been to rationally reconstruct problems from the real world, i.e., we attempted reconstruct systems that exist now or existed in the past.

We have chosen these example problems for several reasons: 1) each has some of the same multi-agent protocol problems as the more complex ATC composite system, but without the attendant jargon and plethora of details, 2) train failures (like ATC failures) are well documented [17], 3)

evaluation functions have the same complex nature, and 4) transportation systems, in general, share a property that is endemic to many other real world composite systems - any new system design must work its way into the existing infrastructure (i.e., a revolutionary approach¹ is not a practical specification development philosophy in most composite system domains). The last two points, evaluation functions and the need to fit a new design into an elaborate existing infrastructure, is taken up in section 4.

2.2: Transformation examples

We could best illustrate our results from these design studies by following several designs through in detail, pointing out how a small number of transformations were used over and over². However, the limited space in this paper will not allow us to do that. Instead, we will briefly describe several of the key transformations used repeatedly in the designs we have produced.

Brinkmanship. This is a standard constraint satisfaction technique. It matches on a conjunctive system constraint, and identifies the brink, i.e., the actions that, if allowed to happen, will push the constraint over the edge. As an example, suppose we have the following constraint taken from figure 3:

```
never: train(T1) & train(T2) & located(T1, L1) & located(T2, L1)
```

The effect of the brinkmanship transformation here would be 1) to identify actions that change a train's location (e.g., a move action), 2) to add a control component (e.g., make train movement a controlled action), and 3) to set up a sub-task to assign an agent to be the controller³. Thus, brinkmanship sets up a goal for subsequent responsibility assignment. Looking at Feather's original development of the elevator, one can see that much of the work exactly this type of goal "jittering" process. For example, Feather introduces a goal/constraint that a passenger must not be in an elevator moving the 'wrong way'. Application of brinkmanship transforms this to a control problem: prevent the passenger's entry into the

1. *Revolutionary* = design from scratch (also known as green-field). *Evolutionary* = work a design in to an existing infrastructure, e.g., existing physical structures, existing laws, existing standards, existing work practices, etc. (also known as brown-field).

2. This is exactly the style used in [9] - a lengthy composite system specification design is presented along with the transformations employed.

3. Clearly there are other strategies/transformations that are possible, e.g., disallow two trains in the system at the same time, make the set of locations of two trains mutually exclusive (e.g., provide two sets of disjoint tracks). These, along with the brinkmanship strategy, are encoded in domain-independent terms.

elevator. Once this goal-jittering step is carried out, control (responsibility) can be assigned to an agent.

Spatial-split. This introduces a standard, multi-agent problem solving protocol. It breaks goal responsibility into two spatially-disjoint pieces¹. A separate agent is assigned to each piece, with responsibility shifting from one agent to the next. Looking back at the brinkmanship problem in the train example, there are actually two actions that we must worry about: trains already under control of the rail line system moving into the same location, and trains entering control of the rail line system (say, from a holding yard). This eventually leads to two actions to control. The spatial-split transformation would suggest that responsibility be split with one agent (i.e., a dispatcher) assigned to entry or "train-in-yard" and a second agent (i.e., an engineer) assigned to movement or "train on line"². As a side-note, there is a drawback that goes along with most of the split-responsibility transformations - a clear handoff protocol must be agreed upon among agents, and inter-agent communication must be reliable. Looking at documentation of train crashes, one finds an alarming rate of train accidents caused by handoff failures [17].

Indirect access. This introduces a standard, multi-agent problem solving protocol, which calls on an agent A to signal another agent B as to the state S of the system. Typically, S is something that B must know to act responsibly, but B does not have direct access to S. In these case, A is asked (given the responsibility) to sense S³ and pass the information along to B. Such inter-agent communication is ubiquitous in transportation systems, e.g., elevators letting passengers know what direction they are heading, station operators signalling trains when the track ahead is clear, air traffic controllers warning a plane that its landing gear has not properly deployed.

Responsibility accumulation. We may assign multiple responsibilities to the same agent. The transformation action is to merge in new responsibilities to those already existing in an agent. The motivation for this transformation is obvious when looking at more complex systems: agents typically play multiple-roles. For instance, train engineers share responsibility for both train progress and train safety goals. Of course, that there is the potential for conflict here

is well documented in case studies of train crashes [17]. In general, responsibility overloading achieves cost savings at the expense of decreased reliability.

The results of our experiments were positive - we were able to identify a set of domain-independent transformations that sufficed to construct a number of real-world designs in each of several transportation domains. At the heart of this set lay a small core of transformations that account for the vast majority of design steps - this core comprises the transformations that we described above. Beyond this core set, we identified transformations that are needed to add components to the design that address agent reliability and motivation issues; the interested reader is referred to [9] for a more thorough discussion.

3: Evaluating composite designs

We argued in the last section that a relatively small number of domain-independent transformations can account for an interesting class of composite system design problems. Thus, if a problem from any domain can be viewed in a composite system light, our domain-independent transformations could construct the space of designs which covers that problem. It does not follow, however, that we have no need of domain knowledge in designing real systems. Viewing CSD as a search process, we need a form of "heuristic function" which can evaluate the designs in the search space and select those which will lead to implementable, reliable, safe, cost-effective systems.

As an example of the kind of evaluation problem which arises in CSD, reconsider the ProtectTrains constraint in figure 3. We could (and did) use our transformations to produce the following two alternative responsibility assignments⁴:

1. Assign ProtectTrains to dispatchers and engineers.
2. Assign ProtectTrains to dispatchers, engineers, and station operators (who controlled track-clear signals).

Both of these showed up in actual train management systems. The first describes pre-1850 train management. It relied on clever scheduling and line-of-sight by engineers to avoid crashes. The second describes most post-1850 train management systems, which employ the notion of protected blocks of track. This is the same basic design that is used today, with computers replacing or augmenting the human agents of the pre-computer era.

Given these two choices 140 years ago, which would we have made? The first leads to a large number of accidents. When the second was introduced, a dramatic drop in

1. Non-spatial splits are also possible, e.g., split by property, split by time, etc.

2. We also have a *single-assignment* transformation that would make a single agent responsible for the entire constraint, i.e., control of both actions.

3. Variations exist where A simply "knows" S - there is no need for a separate sensing operation. For example, a passenger A knows its destination; most elevator systems give A the responsibility of providing that information to the elevator B so it can carry out its responsibility of getting the passenger to their destinations.

4. Of course, we could also generate all other combinations possible.

accidents ensued. Hence, the second seems the best choice. However, the first remained in place long after 1850 on some lines - it appears that the expense of erecting stations, signals and telegraph lines along with paying the station operators outweighed the cost of accidents. In summary, domain-dependent measures of cost and safety are needed to choose among the alternatives generated by domain-independent transformations.

In this section we look at the form evaluation knowledge takes in the transportation problems that we have studied. In reviewing the literature on our example domains, we have identified several classes of domain-specific knowledge needed to make CSD tractable. However, we also note a number of problems integrating this knowledge into CSD which are not addressed by requirements engineering research to date.

3.1: Revolutionary (greenfield) models

Each domain has a collection of specific metrics and models to evaluate a specified component *in isolation*, i.e., as a greenfield system. Figure 3 gives a collection of these for the evaluation of passenger rail system protocols and layouts from [15] [18].

They include:

- Analytic models for estimating capacity of a rail line under a given protocol.
- Statistical models to predict accidents at crossings with different protocols and signals.
- Guidelines for estimating station traffic using embedded transportation systems.

[19] and [12] discuss integration of similar “nonfunctional requirements” into the design of VLSI and algorithms, respectively. We could apply some of the more formal models to the specification directly. For the less formal models, or those which have highly specialized inputs, it will be necessary to rely on a domain expert to map from the artifact’s abstract specification into the terms required by a given model. In addition, many domain-specific evaluation models make assumptions which constrain the implementation of specification constructs. A model for the operating cost of a train signal, for instance, would necessarily identify the technology used to implement that signal. Thus, it remains an open question how and when to apply a given evaluation model in the CSD process.

3.2: Evolutionary (brownfield) models

In the domains we have studied, brownfield constraints prune out many otherwise plausible designs. A principle brownfield constraint is the *operating impact* the artifact may have on its environment. Suppose that in specifying an

elevator system we apply a “split” transformation which divides responsibility among passengers and elevators according to the distance they wish to travel. Passengers will be solely responsible for reaching their destination (via stairs) when they want to go up or down only one floor, while elevators and passengers will be jointly responsible when the distance to travel is greater. Naive application of greenfield metrics for the elevator controller specification, such as cost, performance, and maintainability, might make this seem to be a superior design. This protocol, however, is ruled out for new elevators by local, state, and federal requirements for handicapped access [2].

Figure 3 Evaluating rail and crossing designs.

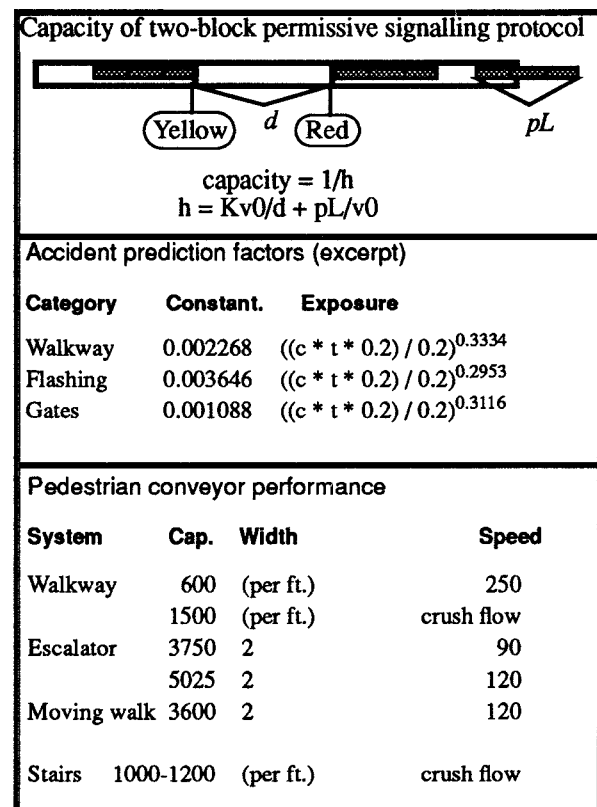
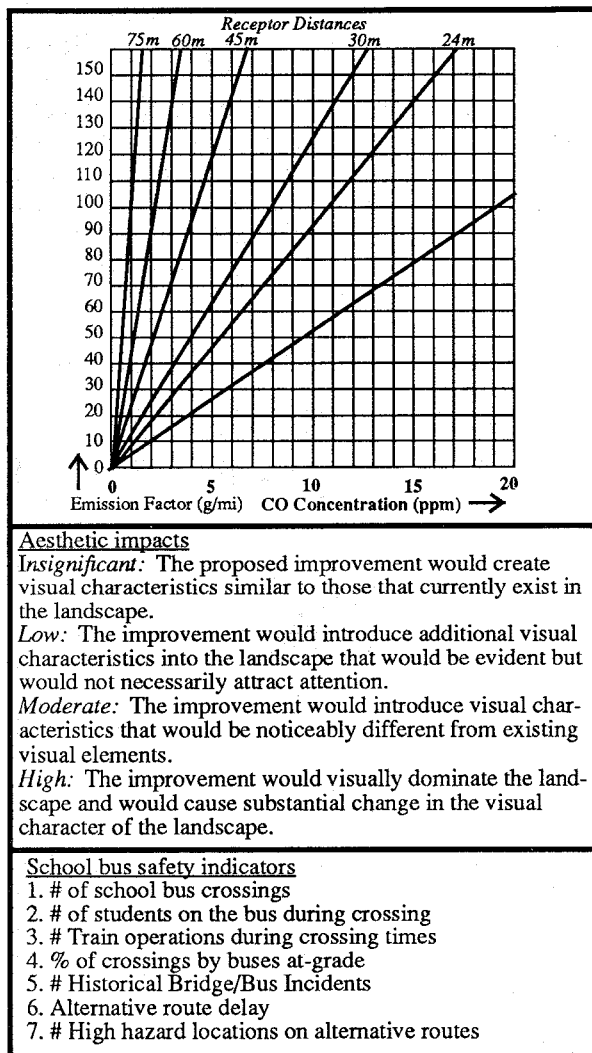


Figure 4 shows a selection of models from [18] for evaluating the impact of rail-highway crossing alternatives on their environment. These include graphical models for computing carbon monoxide emissions, guidelines for evaluating aesthetic impacts, and checklists of special concerns such as emergency vehicle and school bus movements. In general, to genuinely evaluate specification alternatives, we have to consider the written codes and guidelines which have developed in the domain to limit the disruption a system creates in its environment.

Figure 4 Operating impact models.



In addition, the specification process of large-scale composite systems must consider the *transition impact* from standards which may exist in the given domain. Agents, whether software, hardware, or human, must be programmed, designed, or trained to carry out a given set of responsibilities. In a system where a large group of agents have similar responsibilities, it is typically costly to alter these responsibilities. In the rail-highway crossing, for instance, we cannot easily change the protocol by which cars cross railroad tracks "at-grade" (on the same level as the track), even if our new protocol would be a major improvement. The cost of training all potential agents (the drivers) of their new responsibilities is likely to be prohibitive. Even altering the format or behavior of the lights, gates, and gongs at crossings would probably entail unreasonable expense in the short term, since it would require selecting custom hardware and software over the cheaper, mass-produced equipment available to support the

standard design. At present, few requirements engineering models attempt to evaluate transition impacts when examining a system.

4: Summary and future work

We return to our state-based search perspective to discuss the results of our study of CSD. There are two search components to consider: 1) the design operators that construct new states, and 2), the evaluation heuristics that guide the search to acceptable solutions (composite system specifications). Our design operators take the form of transformations on goals and agents. We have been able to produce a small number of powerful transformations that apply across the transportation problems that we have studied, and which could serve as the basis of an interactive design assistant. These transformations are interactive because we lack the formal analytic models (e.g., theorem provers) to guarantee the responsibility-assignment operation of CSD. We have partially plugged this analysis gap with tools like OPIE and the RG tool. We also continue to work on automating our transformations, gradually making them less dependent on human intervention.

It is not surprising that evaluation heuristics are crucial to making composite design search tractable, and that integrating these heuristics into the search appears to be a more difficult problem than defining transformations which generate the search space. On the positive side, we have found numerous formal and informal models for evaluation in the domains we have studied. By studying small but *realistic* composite system problems, we have also begun to identify some important types of evaluation - brownfield and greenfield -- that these models perform, and the kinds of knowledge needed for these model types.

On the negative side, there has been little work in the requirements and specification field to address the issues of integrating greenfield and brownfield models into the design process. One immediate gain would be to integrate existing evaluation models into a tool based on CSD. This is a task we have taken on in a tool we call Critter [9]. Our initial goal is to *informally* catalog the type of models discussed in section 4. Of course, the actual integration of these models into an *automated* search-based design tool is a difficult task, indeed - it requires mapping between multiple ontologies at various levels of formalism. While Farley&Liu have shown that this is possible in non-composite domains [5], we believe much hard work lies ahead to scale their results up.

5: Acknowledgments

Feather has been supported in part by Defense Advanced Research Projects Agency grant No. NCC-2-

520, and in part by Rome Air Development Center contract No. F30602-85-C-0221 and F30602-89-C-0103. All three authors have been supported by NSF grant No. CCR-8804085. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official opinion or policy of DARPA, RADC, NSF, the U.S. Government, or any other person or agency connected with them.

6: References

- [1] Anderson, J. S. & Fickas, S., A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem. *Proceedings: Fifth International Workshop on Software Specification and Design* (Pittsburgh, PA). ACM SIGSOFT Engineering Notes, Volume 14, Number 3 (May 1989).
- [2] ANSI national standard safety code for elevators and escalators, ANSI/ASME A17.1-1987.
- [3] Castro, J., Distributed System Specification using a temporal-causal framework (Ph. D. thesis), Imperial College of Science and Technology and Medicine, University of London, Department of Computing, 1990.
- [4] Dubois, E., Hagelstein, J., A logic of action for goal-oriented elaboration of requirements, in *Proceedings: 5th International Workshop on Software Specification and Design* (Pittsburgh, Pennsylvania, May 19-20, 1989) In *ACM SIGSOFT Engineering Notes* 14(3) (May 1989).
- [5] Farley, A. M., Liu, Z. Y., Shifting Ontological Perspectives in Reasoning about Physical Systems, *Proceedings of the 1990 AAAI Conference*, Boston.
- [6] Feather, M. S., Language Support for the Specification and Development of Composite Systems. *ACM Transactions on Programming Languages and Systems*, 9(2), 198-234.
- [7] Feather, M.S. The evolution of composite system specifications, *Proceedings of the 4th International Workshop on Software Specification and Design*, Monterey, California (USA), April 3-4, 1987.
- [8] Fickas, S., Automating the Transformational Development of Software. *IEEE Transactions of Software Engineering*, 11(11), p. p. 1268-1277.
- [9] Fickas, S., Helm, R., A transformational approach to composite system specification, TR-90-19, CS Dept., U of Oregon, Eugene, Or., 97403
- [10] Huber, P., Jensen, A., Jepsen, L., Jensen, K., Reachability trees for high-level Petri nets, *Theoretical Computer Science* 45 (1986) 262-292.
- [11] *Proceedings of the 4th International Workshop on Software Specification and Design*, Monterey, California, April 3-4, 1987, IEEE Computer Society Press.
- [12] Kant, E., On the efficient synthesis of efficient programs, in Rich, C. and Waters, R. (eds.), *Readings in Artificial Intelligence and Software Engineering*, 285-305, Morgan Kaufmann, 1986. Originally in *Artificial Intelligence* 20, 1983, 253-306.
- [13] Kemmerer, R. A., Testing formal specifications to detect design errors, *IEEE Transactions of Software Engineering* 11(1), p. p. 32-43 (January 1985)
- [14] London, P.E., and Feather, M.S., Implementing specification freedoms, In Rich, C. and Waters, R. (eds.), *Readings in Artificial Intelligence and Software Engineering*, 285-305, Morgan Kaufmann, 1986. Originally in *Science of Computer Programming* 2 (1982) 91-131.
- [15] McGean, T. *Urban transportation technology*. Lexington, MA: D. C. Heath and Company, 1976.
- [16] Reubenstein, H. B. and Waters, R. C., The requirements apprentice: an initial scenario, *Proceedings: 5th International Workshop on Software Specification and Design* (Pittsburgh, Pennsylvania, May 19-20, 1989) In *ACM SIGSOFT Engineering Notes* 14(3) (May 1989).
- [17] Shaw, Robert B. *Down Brakes: a History of Railroad Accidents, Safety Precautions, and Operating Practices in the United States of America*. London, United Kingdom: P. R. Macmillan Limited, 1961.
- [18] Taggart, R. C., Lauria, P., Groat, G., Rees, C., Brick-Turin, A. (1987), "Evaluating grade-separated rail and highway crossing alternatives", National Cooperative Highway Research Program Report 288, Transportation Research Board, National Research Council, June 1987.
- [19] Tong, C., Making routine design possible and robust: research directions, Technical report WP-TR-140, Department of Computer Science, Rutgers University, New Brunswick, NJ 08903.
- [20] Wilbur-Ham, M. C., Numerical Petri Nets - A Guide. Telecom Australia Research Laboratories, Report 7791.
- [21] Zeleny, Milan, *Multiple Criteria Decision Making*. New York: McGraw Hill, 1982.